Inheritance

CSE 114 INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Announcements

Today: Implementation inheritance

Reading assignment for this slide set: my notes

Relationship between classes (objects)

‘Has-a’ relationship (object composition)

- A circle ‘has a’ point as its center.
- A point is composed into a circle.
- A circle is dependent on a point: if the point changes, the circle accordingly will change its location.
- Multiple objects could compose into an object, e.g., two points into a rectangle.

‘Is-a’ relationship (inheritance)

- A student ‘is a’ person. Student class as a subclass of Person class.
- SavingsAccount and CheckingAccount as subclasses of Account class.

- Object is a supertype of any class or interface in Java.

Inheritance (implementation inheritance)

Interface: a mechanism used to provide a 'design' that can be inherited by subclasses.

Inheritance: a mechanism used to provide a 'design and implementation' including fields that can be inherited by subclasses.

See Person.java, Employee.java, Manager.java
(first without inheritance and then with inheritance)

- You can extend these three classes by having them implement the Comparable interface for greater than, less than, or equal to comparisons.

The protected modifier

public: visible by any class

protected: visible by itself and its subclasses
(Rules about protected are more complicated than this, but this is good enough for now)

private: visible only by itself

Shape interface (revisited)

Or, interface vs. inheritance revisited.

- Also will discuss single inheritance vs. multiple inheritance.

Shape.java: this interface contains the common design of all geometric shapes, the behavior part of the shapes!

Shape interface (revisited)

UseShape.java: See how Shape is used in UseShape

- It was used much like a superclass (e.g., Person) was used in relation to a subclass (e.g., Employee), i.e., Shape is superinterface to Circle and Rectangle classes.

A class can extend only one other class in Java. Thus, we say that Java is said to support *single inheritance*. This is really *single implementation inheritance*.

A class can also (in addition to the one class it can extend) implement as many interfaces as it wants to. Thus, we say that Java is said to support *multiple inheritance*. This is really *multiple interface inheritance*.

So, Java supports *single implementation inheritance* but *multiple interface inheritance*.

Summary

How many super class can a Java class have?

One

How many super interfaces can a Java class have?

As many as it wants

How many super interfaces can a Java interface have?

As many as it wants

How does an interface inherit another interface?

Using the ‘extends’ keyword (not ‘implements’)

Array of persons, employees, and managers

If you create an array with its element type Person, we can add Employee instances and Manager instances to the array in addition to Person instances.

- This array is said to be *polymorphic*!

It is acceptable to have these different types of objects in the same array because of the subtyping relationship among these types.

You can see the benefit of subtyping working for us as we write programs using objects that enjoy inheritance!

We have seen examples of this sort using Shape and UseShape. Although there we used an interface and classes, classes among themselves work in the same way as long as there are supertype subtype relationships established among them.

Object-oriented vs. function-oriented

‘**Function-oriented**’: function is the ‘master’ and objects or data are just being passed around between functions to be operated on. For example:

- **foo(a, b)**
- We did quite a bit of programming this way early in the semester before we started discussing objects. The static methods that we used then were like this although static methods themselves were part of an object, not a dynamic object but a static object. We are really talking about dynamic objects in this context.

Object-oriented: object is the ‘master’ and it ‘owns’ its data (fields) and behavior (methods):

- **a.foo(b)**
- Here **a** is really a dynamic object; **b** may or may not be an object.

Software development process

1. Understand the problem in terms of its requirements
2. Design the software
   - What classes
   - What fields/methods in each class
     - What is static, dynamic?
     - What is private, protected, public?
   - Inheritance and composition relationships
3. Implement the design
4. Test the implementation

Often iterating over the steps as you debug your implementation and/or design.

Test-first development

You should try to see how your design will be used by use code later. That is, see how UseX will be using your X.

Try to write some use code in the main of UseX to see how X will be used. That greatly helps you understand what X should provide.

In fact, you should write the use code (UseX) first before you write the base code (X).

We call this type of development *test first development*.

We did quite a bit of that this semester!

Something to consider

A variable of supertype can hold a subtype object

A variable of subtype cannot hold a supertype object. Why not?